软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007091]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于最短路径序列化图的域内路由保护算法*

耿海军[1,2], 胡睿乾[2], 胡治国[2,3], 尹 霞[4]

[1](山西大学 自动化与软件学院, 山西 太原 030006)

[2](山西大学 计算机与信息技术学院, 山西 太原 030006)

[3](嵌入式系统与服务计算教育部重点实验室 (同济大学), 上海 200092)

[4](清华大学 计算机科学与技术系, 北京 100084)

通信作者: 耿海军, E-mail: genghaijun@sxu.edu.cn

摘 要: 互联网服务提供商采用路由保护算法来满足实时性、低时延和高可用应用的需求. 然而已有路由保护算法存在下面 3 个方面的问题: (1) 在不改变传统路由协议转发机制的前提下, 故障保护率普遍较低; (2) 为了追求较高的故障保护率, 通常需要改变传统路由协议的转发机制, 实际部署难度较大; (3) 无法同时利用最优下一跳和备份下一跳, 从而导致网络负载均衡能力较差. 针对上述 3 个问题, 提出一种基于最短路径序列化图的路由保护算法, 所提算法不需要改变转发机制、支持增量部署、同时使用最优下一跳和备份下一跳不会出现路由环路、并且具有较高的故障保护率. 所提算法主要包括下面两个步骤: (1) 为每个节点计算一个序号, 构造最短路径正序化图; (2) 利用最短路径正序化图和反序搜索规则构造最短路径序列化图, 在此基础上根据备份下一跳计算规则计算节点对之间的备份下一跳集合. 在真实和模拟网络拓扑上进行测试, 实验结果表明, 与其他路由保护算法相比, 所提算法在平均备份下一跳数量、故障保护率和路径拉伸度 3 个指标方面均具有显著的优势.

关键词: 网络故障; 路由保护; 最短路径序列化图; 故障保护率; 路径拉伸度

中图法分类号: TP393

中文引用格式: 耿海军, 胡睿乾, 胡治国, 尹霞. 基于最短路径序列化图的域内路由保护算法. 软件学报. http://www.jos.org.cn/1000-9825/7091.htm

英文引用格式: Geng HJ, Hu RQ, Hu ZG, Yin X. Intra-domain Routing Protection Algorithm Based on Shortest Path Serialization Graph. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7091.htm

Intra-domain Routing Protection Algorithm Based on Shortest Path Serialization Graph

GENG Hai-Jun[1,2], HU Rui-Qian[2], HU Zhi-Guo[2,3], YIN Xia[4]

[1](School of Automation and Software Engineering, Shanxi University, Taiyuan 030006, China)

[2](School of Computer and Information Technology, Shanxi University, Taiyuan 030006, China)

[3](The Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education, Shanghai 200092, China)

[4](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: Internet service providers employ routing protection algorithms to meet real-time, low-latency, and high-availability application needs. However, existing routing protection algorithms have the following three problems. (1) The failure protection ratio is generally low under the premise of not changing the traditional routing protocol forwarding mechanism. (2) The traditional routing protocol forwarding mechanism should be changed to pursue a high failure protection ratio, which is difficult to deploy in practice. (3) The optimal next hop and backup next hop cannot be utilized simultaneously, which causes poor network load balancing capability. For the three problems, this

* 基金项目: 山西省应用基础研究计划 (20210302123444, 20210302123455); 山西省高等学校科技创新项目 (2022L002); 中国高校产学研创新基金 (2021FNA02009); 同济大学嵌入式系统与服务计算教育部重点实验室开放课题 (ESSCKF 2021-04); 山西省重点研发计划 (202202020101004); 国家自然科学基金 (61702315); 国家重点研发计划 (2018YFB1800401)
收稿时间: 2023-06-02; 修改时间: 2023-08-02, 2023-09-28; 采用时间: 2023-11-16; jos 在线出版时间: 2024-03-27

study proposes a routing protection algorithm based on the shortest path serialization graph, which does not need to change the forwarding mechanism, supports incremental deployment and adopts both optimal next hop and backup next hop without routing loops, with a high failure protection ratio. The proposed algorithm mainly includes the following two steps. (1) A sequence number for each node is calculated, and the shortest path sequencing graph is generated. (2) The shortest path serialization graph is generated based on the node sequence number and reverse order search rules, and the next hop set between node pairs is calculated according to the backup next hop calculation rules. Tests on real and simulated network topologies show that the proposed scheme has significant advantages over other routing protection schemes in the average number of backup next hops, failure protection ratio, and path stretch.

Key words: network failure; routing protection; shortest path serialization graph; failure protection ratio; path stretch

随着互联网的飞速发展和扩张, 自治系统的规模和数量急剧增长, 给域内路由带来了许多迫切需要解决的关键问题, 其中路由可用性问题[1,2]尤为突出. 研究表明, 网络故障频繁出现且不可避免. 当这些故障发生时, 传统的路由协议通常无法在 150 ms 内完成收敛, 难以满足实时应用[3,4]如 IP 语音、股票在线交易、远程手术、视频流媒体和即时通信等对网络收敛时间的苛刻需求. Akamai 的一项研究[5]更是指出网站加载时间每延迟 100 ms 就会导致销售额大幅下降; 语音通信和游戏的可容忍延迟分别小于 150 ms 和 80 ms. 因此, 当网络故障[6,7]出现时, 若没有有效的应对机制, 互联网服务提供商 (Internet service provider, ISP) 将无法兑现承诺的服务质量, 进而影响其声誉和收益, 甚至可能引发严重的事故. 例如, 过去的网络故障已导致亚洲地区出现大范围的互联网中断, 给互联网行业带来巨大的经济损失, 影响了数十万名航空乘客的行程, 甚至导致应急网络被迫中断. 因此, 提高域内路由可用性[8,9]已成为互联网发展亟待解决的关键科学问题.

学术界和工业界广泛应用路由保护算法来应对网络中频繁发生的故障. 然而, 目前路由保护算法面临的主要挑战包括: 首先, 基于控制层面的路由恢复算法需要等待路由协议重新收敛完成后才能正常转发报文, 而在路由协议收敛过程中, 大量报文将被丢弃. 其次, 基于数据层面的路由保护算法通常是预先计算出节点对之间的备份路径, 在路由收敛过程中利用这些备份路径转发受故障影响的报文. 当路由重新收敛完成后, 在新的拓扑中重新计算备份路径. 但是, 一些基于数据层面的路由保护算法, 如 Not-Via[10], 段路由 (segment routing, SR)[11], 多配置路由 (multiple routing configurations, MRC)[12], 故障携带路由 (failure carrying packet, FCP)[13], 为了达到提高路由可用性的目的, 往往需要进行复杂的配置. 这些复杂的配置主要包括修改报文的转发方式和增加额外的转发机制来完成报文的转发, 这无疑增加了网络开销, 导致路由保护算法在实际部署中面临困难[14,15].

本文将致力于解决目前路由保护算法存在的上述问题. 具体贡献如下.

(1) 针对最短路径正序化图[16]在路由保护方面的局限性, 我们首次提出了节点反序关系、反序搜索规则和备份下一跳计算规则. 基于这些规则, 我们设计了全新的基于最短路径序列化图的域内路由保护算法. 该算法在转发报文时, 能同时利用最优下一跳和备份下一跳, 且有效避免了路由环路的出现.

(2) 本文提出的算法无需改变当前互联网广泛使用的路由协议的转发方式, 因此实现简便, 便于实际部署. 此外, 该算法无需进行复杂的配置. 当数据报文在路由转发过程中遇到故障时, 路由器在接收到该报文后, 会立即将其转发至预先计算出的备份下一跳节点, 无需等待路由重新收敛, 从而大大提高了报文正确转发的概率.

(3) 我们在大量真实和模拟拓扑中对本文提出的算法进行了性能测试. 实验结果表明, 该算法在平均备份下一跳数量、故障保护率和路径拉伸度等多个关键指标上均表现出优越的性能.

本文第 1 节介绍路由保护的相关工作. 第 2 节描述网络模型和问题. 第 3 节探讨构造最短路径序列化图的方法. 第 4 节详述算法细节. 第 5 节将本文算法与其他算法进行实验对比, 并分析实验结果. 最后, 第 6 节对全文进行总结与展望.

1 路由保护相关工作

基于数据层面的路由保护算法主要可以分为基于逐跳转发策略和基于非逐跳转发策略两大类. 基于逐跳转发策略的路由保护算法的转发方式和目前部署的路由协议的转发方式是一致的, 不需要借助辅助转发机制的协助. 然而, 基于非逐跳转发策略的路由保护算法的转发方式需要增加额外的转发机制来完成报文的转发, 这种转发方

式明显地增加了网络开销.

基于逐跳转发策略的路由保护算法主要包括等价多路径路由 (equal cost multiple paths, ECMP), loop free alternates (LFA)[17−19], MARA-MA 和 MARA-SPE[20]. ECMP 是业界最早采用的一种简单的路由保护算法, 互联网部署的 OSPF 和 IS-IS 协议都支持 ECMP, 但是 ECMP 的故障保护率过低[21]. 针对 ECMP 故障保护率低的问题, 国际互联网工程任务组 (Internet engineering task force, IETF) 发布了快速重路由的框架, 在该框架的基础上提出了无环路路由 (loop-free alternates, LFA). LFA 包括链路保护条件 (link protection condition, LFC)、单结点保护条件 (node protection condition, NPC) 和下游规则 (downstream criterion, DC). DC 是一种经典的无环路规则, 利用该规则计算节点可以选择比自己到达目的地址更近的邻居节点作为计算节点到达目的节点的备份下一跳. 虽然 DC 简单且容易部署, 但是 DC 的故障保护率依旧比较低. MARA-MA、MARA-SPE 和 RPP-MTMAX[16]可以进一步提高 DC 规则的故障保护率. 这 3 种方法都是通过构造以目的地址为根的有向无环图来计算源节点到目的节点的备份下一跳, MARA-MA 以最大化最小连通性为目标构造有向无环图, MARA-SPE 在 MARA-MA 基础上加入了有向无环图必须包含以目的为根的最短路径树的限制条件. RPP-MTMAX 以最大化节点对之间至少拥有一个备份下一跳的节点对数量为目标构造有向无环图. 上面介绍的算法中, ECMP、DC、MARA-MA、MARA-SPE 和 RPP-MTMAX 本质上都是以目的为根构造了一个偏序关系, 即每一个备份下一跳到达目的地址总是越来越近, 这样既可以保证转发路径无环路又可以计算出多个备份下一跳. 但是这些方法仅仅考虑了以目的为根的节点对之间的正向偏序关系, 而忽略了节点对之间的反向偏序关系.

基于非逐跳转发策略的路由保护算法主要包括 Not-Via、SR、MRC 和 FCP. Not-Via 算法使用特殊地址 Not-Via 显示说明网络中的故障, 从而在转发报文的过程中有效避开该故障. 当报文在转发过程中遇到故障时, 该报文将会被封装成特殊形式的报文, 然后转发到合适的中转节点, 最后中转节点对该报文解封装, 并且按照最短路径转发该报文. 基于 SR 的路由保护算法将网络的路径分成了一个一个的小段, 然后为这些段和网络结点分配 Segment Routing ID, 通过对这些 Segment Routing ID 进行有序的排列, 就可以生成一条完整的转发路径. MRC 通过改变网络中链路的代价计算出多个配置图, 这些不同配置图可以应对网络中出现的不同故障从而达到提高路由可用性的目的, 但是 MRC 部署开销较大. FCP 的核心思路如下: 首先将报文在转发过程中遇到的故障存到其头部, 然后根据该头部信息更新网络拓扑结构, 最后利用新的网路拓扑结构计算新的最短路径. FCP 可以应对网络中的故障问题, 但是该方法需要改变路由协议, 部署难度较大. 上述算法都需要修改互联网部署的路由协议的转发方式, 甚至修改报文头部的控制字段, 因此实现难度较大.

2 网络模型和问题描述

在本节中, 我们将阐述网络模型和问题描述. 为了方便理解本文中所使用的定义、定理和算法, 表 1 列出了一些符号和它们表示的含义.

表 1 符号说明

符号	含义	符号	含义
$G=(V,E,W)$	网络拓扑结构	$len(priority\_queue(Q))$	队列 Q 的长度
$routerID(v)$	节点 v 的路由器 ID	$ps(L_d)$	表示节点正序关系
$<u,v>$	由节点 u 指向节点 v 的单向边	$rs(L_d)$	表示节点反序关系
(u,v)	节点 u 与节点 v 之间的双向边	$x.sequence$	节点 x 的序号
$backn(a,d)$	节点 a 到节点 d 的备份下一跳集合	$hop_{[G]}^{\leqslant\lvert V\rvert}(v)$	拓扑中节点 v 拥有的下一跳节点数量
G'_d	目的地址为 d 的最短路径正序化图	$R(G'_d)$	表示在拓扑 G'_d 上运用反序搜索规则
G''_d	目的地址为 d 的最短路径序列化图	$P(a,b)=0$	节点 a, b 之间的可达性为0

2.1 网络模型

网络模型可以抽象为一个无向连通图 $G=(V,E,W)$, 在 G 中用 V 来指代路由器 (节点) 集合, 用 E 表示相邻路

由器之间的链路 (边) 集合, W 表示链路的代价集合, 代价可以是时延、带宽和丢包率等. 当代价为时延、丢包率时, 代价的数值越小说明链路拥有更高的优先级, 然而当代价为带宽时, 代价的数值越大说明链路拥有更高的优先级. 对于拓扑图中的任意节点 $v \in V$, 用 $routerID(v)$ 表示节点 v 的路由器 ID, 路由器 ID 越小, 其优先级越高; $nei(v)$ 表示节点 v 的邻居节点集合. 如果 $u \in nei(v)$, (u,v) 表示节点 u 与节点 v 之间的双向边, $<u,v>$ 表示节点 u 指向节点 v 的单向边. T_d 指以目的节点 d 为根的反向最短路径树, 节点 a 到节点 d 的最短路径中的第 1 个节点为节点 a 到节点 d 的最优下一跳节点 (最优下一跳). $backn(a,d)$ 表示节点 a 到节点 d 的备份下一跳集合. 节点的序号是指通过某种规则给节点编号, 比如目的节点 d 的序号为 0, 除了目的节点的序号为 0 以外, 其余节点的序号取值范围在 $1–|V|-1$ 之间.

在拓扑图中, 如果给每个节点进行编号, 对于存在邻居关系的两个节点 x 和 y, 如果 x 的序号大于 y 的序号, 则把节点 x 和 y 之间的关系称为节点正序关系, 反之, 如果 x 的序号小于 y 的序号, 则把节点 x 和 y 之间的关系称为节点反序关系. 节点正序关系和节点反序关系都称为节点序列关系. 容易证明, 如果节点 x 和 y 之间符合正序关系, 则节点 x 可以把报文转发给 y 而不会出现路由环路. 相反, 如果节点 x 和 y 之间符合反序关系, 则节点 x 不一定可以把报文转发给 y, 因为这样转发将可能会出现路由环路.

定义 1. 最短路径正序化图. 给定拓扑图 $G=(V,E,W)$ 和反向最短路径树 T_d , 根据节点选择规则为每个节点赋予序号. 对于任意节点 $u \in V$, 如果 $<u,v> \notin T_d$, $v \in nei(u)$ 并且节点 u 和节点 v 满足正序关系, 则将边 $<u,v>$ 加入到 T_d 中. 将按照这种方式构造的拓扑称为最短路径正序化图. 用符号表示为 $G'_d=(V,E',W)$, 简写为 G'_d.

定义 2. 最短路径序列化图. 给定最短路径正序化图 G'_d , 对于任意节点 $u \in V$, 如果 $<u,v> \notin G'_v$, $v \in nei(u)$, 节点 u 和节点 v 满足反序关系, 并且二者之间符合反序搜索规则, 则将边 $<u,v>$ 加入到 G'_d 中. 将按照这种方式构造的拓扑称为最短路径序列化图, 用符号表示为 $G''_d=(V,E',W)$, 简写为 G''_d.

本文将在第 3.1 节和第 3.2 节分别介绍节点选择规则和反序搜索规则. 由以上定义可知, 反向最短路径树 T_d 是 G'_d 的子集, 同时也是 G''_d 的子集, G'_d 是 G''_d 的子集, 三者之间的关系可以表示为:

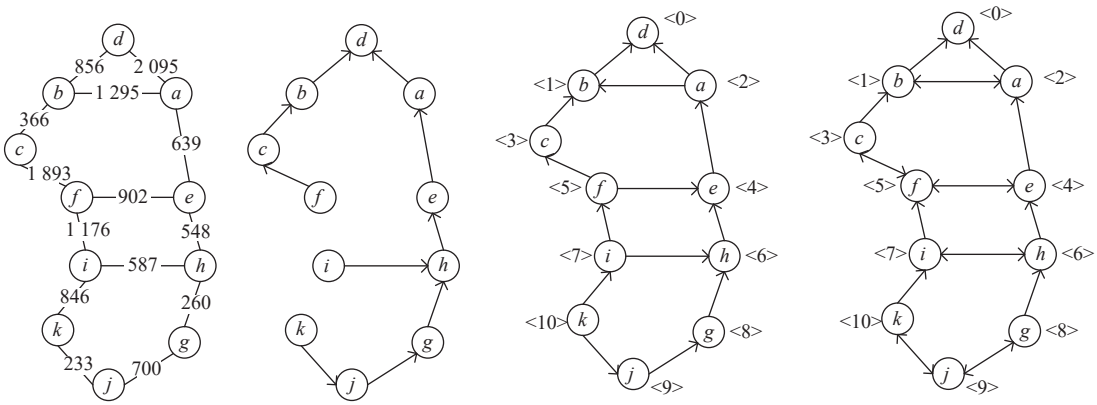
$$T_d \subseteq G'_d,\ T_d \subseteq G''_d,\ G'_d \subseteq G''_d \tag{1}$$

定义 3. 备份下一跳计算规则. 在 G'_d 和 G''_d 中, 对于任意节点 $u \in V$, $v \in nei(u)$ 且节点 v 不是节点 u 到目的 d 的最优下一跳, 如果 $<u,v> \in G'_v$ 或者 $<u,v> \in G''_v$ 成立, 则节点 v 可以作为节点 u 到目的 d 的备份下一跳, 即: $backn(u,d)=\{v\}$.

下面通过一个简单例子来解释上面的定义, 假设目的节点为 d. 图 1(a) 表示 11 个节点 14 条边的拓扑结构, 边上的数值代表链路的代价, 图 1(b) 是该拓扑图对应的反向最短路径树, 图 1(c) 是该拓扑图对应的最短路径正序化图, 图 1(d) 是该拓扑图对应的最短路径序列化图. 节点 d 的序号为 0, 节点 b 的序号为 1, 对于节点 b, 其邻居节点 d 的序号比 b 小, 则节点 b 和节点 d 之间构成节点正序关系; 其邻居节点 c 的序号比 b 大, 则节点 b 和节点 c 之间构成节点反序关系. 节点 e 是节点 f 的邻居节点, 在图 1(b) 中节点 e 不是节点 f 到目的 d 的最优下一跳, 在图 1(c) 中 $<f,e> \in G'_d$, 根据备份下一跳选择规则可知节点 e 可以作为节点 f 到目的 d 的备份下一跳, 即: $backn(f,d)=\{e\}$. 同理, 节点 k 和节点 j 互为邻居节点, 在图 1(b) 中节点 k 不是节点 j 到目的 d 的最优下一跳, 在图 1(d) 中 $<j,k> \in G''_d$, 因此可以得到 $backn(j,d)=\{k\}$. 从图 1 可知, 最短路径序列化图 G''_d 是将最短路径正序化图 G'_d 中的部分单向边修改为双向边, 但是并不是所有的单向边都可以修改为双向边, 这是因为如果随意将 G'_d 中的单向边修改为双向边, 报文在转发的过程中可能会出现路由环路. 比如在图 1(d) 中, 如果将节点 a 和 e 之间的单向边修改为双向边, 这时会产生 $a \to e \to f \to c \to b \to a$ 路由环路. 因此本文将重点讨论如何构造最短路径正序化图, 最短路径序列化图, 分析为什么利用备份下一跳计算规则计算出的路径没有路由环路.

2.2 问题描述

本文要解决的问题可以描述为: 给定网络拓扑图 $G=(V,E,W)$ 和反向最短路径树 T_d , 如何构造最短路径序列化图, 从而使得网络拓扑中节点拥有的备份下一跳数量最大化, 同时最小化无备份下一跳的节点数量. 该问题可以表示为一个线性规划模型 (linear programming model, LPM), 即:



(a) 网络拓扑 G (b) 反向最短路径树 T_d (c) 最短路径正序化图 G'_d (d) 最短路径序列化图 G''_d

图 1 网络拓扑、反向最短路径树、最短路径正序化图和最短路径序列化图

$$\max\ hop_{[G]}^{\leqslant|V|}(d) \tag{2}$$

$$\min\ len(priority\_queue(Q)) \tag{3}$$

s.t.

$$\forall G \neq \varnothing \in Undirected\ connected\ graph(UCG) \tag{4}$$

$$asn = \{y|\ <x,y> \in T_d, y \in [V - L_d], w \in L_d\} \tag{5}$$

$$\sum_{len(L_d)\to 0}^{len(L_d)\to N(V)} \sum_{node\in\Upsilon(asn)} L_d \cup \{node\} \tag{6}$$

$$ps(x,y) = \{x,y \in L_d \wedge y.sequence < x.sequence\} \tag{7}$$

$$rs(x,y) = \{x,y \in L_d \wedge x.sequence < y.sequence\} \tag{8}$$

$$G'_d = T_d + ps(L_d) \tag{9}$$

$$single\_jump = single\_jump \bigcup_{x\in V} \{|backn(x,d)| = 1\} \tag{10}$$

$$G''_d = G'_d + R(G'_d) \tag{11}$$

接下来我们详细解释该 LPM 问题, 其中公式 (2) 和公式 (3) 是目标函数, 公式 (2) 中 $\leqslant|V|$ 表示经过不大于 $|V|$ 次迭代之后, 图 G 中所有节点拥有的下一跳节点数量最大化. 同时, 使存放只有最优下一跳节点的优先队列 Q 的长度最小 (公式 (3)). 公式 (4)–公式 (11) 是约束条件. 公式 (4) 说明拓扑图 G 是一个无向连通图 (UCG) 不是一个空图, 公式 (5) 和公式 (6) 是后文将详细介绍的备选搜索节点和节点选择规则, 这两个公式中的 L_d 是指在遍历 T_d 的过程中已经被遍历节点的集合, 该集合是动态变化的, 初始时 $L_d = \{d\}$. $[V - L_d]$ 是指节点集合减去集合 L_d 后剩余节点构成的集合. 公式 (6) 中最里层式子代表将节点加入到输出集合中, $\sum_{node\in\Upsilon(asn)}$ 表示遍历备选搜索节点集合 $\Upsilon(asn)$ 中的节点, 外层 $\sum_{len(L_d)\to 0}^{len(L_d)\to|V|}$ 代表输出集合 L_d 从空集合累加节点, 直到将全部节点加入到输出集合 L_d , 此时 L_d 的长度等于图中节点个数, 且对于每个节点的加入, 都需要进行里层遍历计算. 公式 (7) 和公式 (8) 是节点序列关系, 在正序关系中, 节点 x 的序号 ($sequence$) 大于节点 y 的序号, 在反序关系中节点 y 的序号大于 x 节点的序号. 公式 (9) 表示最短路径正序化图. 公式 (10) 是寻找图 G 中只有最优下一跳的节点 x, 然后将它们依次到集合 $single\_jump$ 中, $single\_jump$ 是节点集合 V 的子集. 公式 (11) 表示最终得到的最短路径序列化图.

3 构造最短路径序列化图的方法

下面是构造最短路径序列化图包括的 3 个步骤.

步骤 1. 在拓扑图 G 以及反向最短路径树 T_d 的基础上, 根据节点选择规则给每个节点赋予序号, 构造最短路径正序化图.

步骤 2. 在步骤一的基础上, 根据反序搜索规则构造最短路径序列化图.

步骤 3. 通过最终得到的最短路径序列化图, 利用备份下一跳计算规则计算节点的备份下一跳.

在本节中, 我们将分别介绍构造最短路径正序化图和构造最短路径序列化图所需要的理论知识和关键定义.

3.1 构造最短路径正序化图

定义 4. 备选搜索节点. 给定反向最短路径树 T_d, 节点 $x,y \in V$, 从 $[V-L_d]$ 集合中选择新的节点作为备选搜索节点, 应满足如下公式:

$$asn = \{y| <y,x> \in T_d, y \in [V-L_d], x \in L_d\} \tag{12}$$

在计算最短路径正序化图之前, 需要先找到图 G 中的备选搜索节点 (alternative search nodes, *asn*), 选择备选搜索节点是为给 L_d 集合中加入新的节点做准备. 在公式 (12) 中, 节点 x 在输出集合 L_d 中, 节点 y 在 $[V-L_d]$ 集合中, 它们组成的边满足在反向最短路径树上, 那么节点 y 会从 $[V-L_d]$ 加入到 L_d 集合中, 这样的节点就叫做备选搜索节点, 它们组成的集合称为备选搜索节点集合 $\Upsilon(asn)$.

定义 5. 节点选择规则. 如果 $\Upsilon(asn)$ 中仅有一个节点 n 与 L_d 集合相连边数等于 2, 则将其加入到输出集合中, 如果存在相连边数相同的节点, 则选择路由器 ID 值最小的节点加入到输出集合中; 否则, 选择与 L_d 相连边数最多的节点加入到输出集合中, 如果存在相连边数相同的节点, 则选择路由器 ID 值最小的节点加入到输出集合中.

算法 1 描述了构造最短路径正序化图的过程. 算法的输入为网络拓扑结构 G 和目的节点 d, 算法的输出为最短路径正序化图 G'_d. 将目的节点 d 的序号设置为 0, 初始化 L_d 为 d, 构造以目的节点 d 为根的反向最短路径树 (第 1–3 行). 下面是循环过程, 在每一次循环中根据备选搜索节点和节点选择规则选择一个节点 y 加入到 L_d, 并且更新节点 y 的序号 (第 5–11 行). 对于任意节点 $x \in V/\{d\}$, 如果 $y \in nei(x)$ 并且节点 x 和节点 y 满足正序关系, 则将边 $<x,y>$ 加入到 T_d 中 (第 12–18 行). 最后返回最短路径正序化图 G'_d (第 19 行).

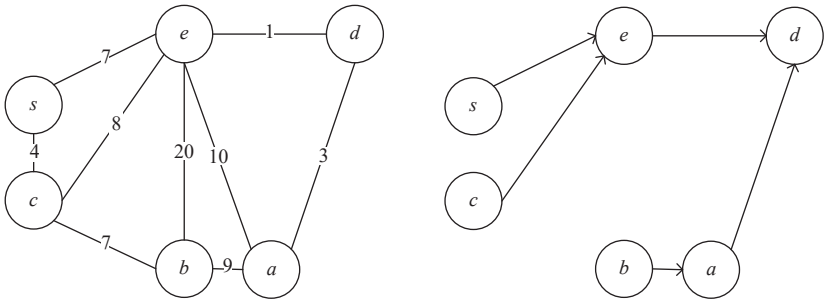
算法 1. SPPOD.

输入: $G=(V,E)$, 目的地址 d;
输出: G'_d.

1. $d.sequence \leftarrow 0$
2. $L_d = \{d\}$
3. Compute T_d
4. $i \leftarrow 1$
5. **While** $i \leqslant |V|-1$ **do**
6. $ans \leftarrow \{y \in V \backslash L_d \,|\, \forall (x,y) \in T_d, x \in L_d\}$
7. select a node $y \in ans$
8. $y.sequence \leftarrow i$
9. $L_d \leftarrow L_d \cup \{y\}$
10. $i \leftarrow i+1$
11. **EndWhile**
12. **Foreach** $x \in V \backslash \{d\}$ **do**
13. **Foreach** $y \in nei$ **do**

14. **If** $y.sequence < x.sequence$ **then**
15. $G'_d \leftarrow T_d \cup < x,y >$
16. **EndIf**
17. **EndFor**
18. **EndFor**
19. **Return** G'_d

下面通过一个例子来说明构造最短路径正序化图的过程. 图 2(a) 所示 6 个节点和 8 条边的网络拓扑结构. 图 2(b) 是该拓扑对应的反向最短路径树拓扑 T_d. 开始将节点 d 加入到输出集合 L_d 中, 其余节点 $[a,b,c,s,e]$ 在 $[V-L_d]$ 集合中. 在图 2(b) 中, 与节点 d 相连且在反向最短路径树 T_d 上的节点是 a, e, 则节点 a, e 是定义 4 中的备选搜索节点, 它们组成的集合是备选搜索节点集合 $\Upsilon(asn)$. 根据节点选择规则可知, a, e 与集合 L_d 相连的边的数量均为 1, 节点 a 的路由器 ID 小于节点 e 的路由器 ID, 故选择节点 a, 并将其加入到输出集合 L_d 中 ($L_d = [d,a]$). 这时再搜索节点, 找到备选搜索节点 e, b, 它们都是在反向最短路径树 T_d 上且与输出集合 L_d 相连, 在原始拓扑 G 中, e 与输出序列 L_d 相连两条边, b 与 L_d 相连一条边, 根据节点选择规则可知, 下一步选择节点 e 加入到输出集合 L_d 中. 依次类推, 可得最后的输出集合 $L_d = [d,a,e,b,c,s]$. 根据输出集合 L_d 中加入节点的顺序, 给节点赋予相应序号, 可得到拓扑 G 的最短路径正序化图 G'_d 如图 3 所示.



(a) 原始拓扑 G (b) 拓扑 G 的反向最短路径树 T_d

图 2 原始拓扑 G 与反向最短路径树拓扑 T_d

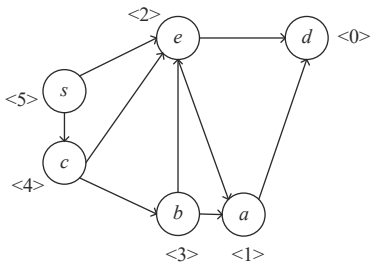


图 3 最短路径正序化图 G'_d

根据上述最短路径正序化图 G'_v 可得到每个节点的序号, 节点 d 的序号为 0, 节点 a 的序号为 1, 节点 b 的序号为 3, 节点 c 的序号为 4, 节点 s 的序号为 5, 节点 e 的序号为 2, 即 $\{d:0, a:1, b:3, c:4, s:5, e:2\}$.

3.2 构造最短路径序列化图

下面介绍反序搜索规则, 并证明利用备份下一跳规则计算出的路径不会出现路由环路.

定理 1. 反序搜索规则 $R(G'_d)$. 给定最短路径正序化图 G'_d, 节点 a 和节点 b 互为邻居关系, $< b,a > \in G'_d$ 和 $< a,b > \notin G'_d$, 将 a 和 b 之间的边修改为双向边以后, 如果 (1) $P(b,a)=0$, (2) $P(a,a)=0$, (3) $P(b,d)=1$ 同时成立, 则

可以将 a 和 b 之间的边修改为双向边, 报文在转发的过程中不会出现路由环路, 即 $<<a,b>\in G_d''$; 否则将不能将 a 和 b 之间的边修改为双向边. 其中 $P(b,a)=0$ 表示如果不考虑 $<b,a>$ 之间相连的边的情况下, 节点 b 和节点 a 之间不存在其他路径; $P(a,a)=0$ 表示节点 a 不存在其他路径到达自身; $P(b,d)=1$ 表示节点 b 和节点 d 之间存在其他路径.

证明: 首先利用反证法证明条件 (1), 不包括原来从节点 b 到节点 a 的单向边, 如果在图 G_d' 中存在一条从 b 到 a 的路径, 假设数据报可以通过第 3 个节点 c 到达 a, 即 $b\to c\to a$, b 到 a 的可达性为 1, 表示为 $P(b,a)=1$. 如果将 a 和 b 之间的边修改为双向边, 则数据报又可以从 a 到达 b, 即 $b\to c\to a\to b$, 这样图中就会产生环路.

然后利用反证法证明条件 (2), 如果将 a 和 b 之间的边修改为双向边, 节点 a 存在一条到自身的路径, 则数据报从 a 到达 b, 这样图中就会产生环路.

最后证明当 $P(b,d)=1$, 如果改变 a 和 b 之间的边为双向边, 则节点 a 一定可以找到一条到达目的 d 的无环路径. 如果将 a 和 b 之间的边为双向边, 则节点 a 可以将数据报转发给 b, 根据条件 (1) 和条件 (2) 和 $P(b,d)=1$ 可知, 节点 $P(a,d)=1$. 综上所述, 该定理成立.

从图 3 的例子可以看出, 除去节点 a 到达目的节点 d 没有备份下一跳以外, 其余节点都有备份下一跳.下面将通过反序搜索规则为节点 a 计算备份下一跳. 节点 a 和节点 b 互为邻居关系, $<b,a>\in G_d'$, $<a,b>\notin G_d'$, 如果将 a 和 b 之间的边修改为双向边, 则 $P(a,a)=1$, 因此 a 和 b 之间的边不能修改为双向边, 即 $<a,b>\notin G_d''$. 节点 a 和节点 e 互为邻居关系, $<e,a>\in G_d'$, $<a,e>\notin G_d'$, $P(e,d)=1$, 如果将 a 和 e 之间的边修改为双向边, $P(e,a)=0$, $P(a,a)=0$, 根据上述定理可知, a 和 e 之间的边可以修改为双向边, 即 $<a,e>\in G_d''$. 图 4 为最终的最短路径序列化图 G_d''. 至此, 在该网络拓扑中, 以节点 d 作为目的节点时, 所有节点都可以计算出备份下一跳.

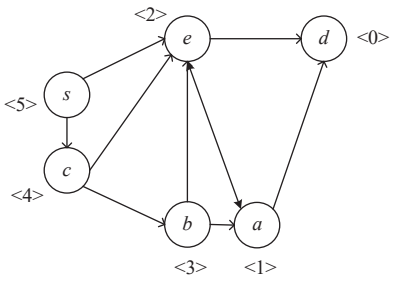


图 4 最短路径序列化图 G_d'' (a, e 之间的边转变为双向边)

定理 2. 在网络拓扑中, 对于任意节点 $u\in V$, 如果节点 u 根据备份下一跳计算规则选择其到达目的地址的备份下一跳, 则节点 u 到目的节点 d 的路径不会出现路由环路.

证明: 在网络拓扑中, 对于任意节点 $u\in V$, 节点 u 计算的备份下一跳有两种情况: (1) 所有节点和其备份下一跳满足正序关系 (2) 节点和其备份下一跳满足正序关系和反序关系. 下面分为两种情况来证明该定理.

情况 (1): 如果所有节点和其备份下一跳满足正序关系, 则备份下一跳到达目的的序号越来越小, 因此转发路径不会出现路由环路.

情况 (2): 如果所有节点和其备份下一跳满足正序关系和反序关系, 由情况 (1) 可知, 当二者之间是正序关系时, 转发路径不会出现路由环路; 当二者之间是反序关系时, 根据定理 1 可知, 转发路径也不会出现路由环路.

根据情况 (1) 和情况 (2) 可知, 该定理成立.

4 算 法

基于最短路径序列化图的域内路由保护算法包含 3 个算法, 它们分别是 LtS-NETDeg 算法、StL-NETDeg 算法和 Reduce-NodPri 算法. 这 3 个算法的核心思路如下: (1) 首先, 构造最短路径正序化图, 为每个网络拓扑中的节点分配一个唯一的序号; (2) 接着, 计算出只具有最优下一跳的节点, 并将它们存储在一个优先级队列中; (3) 随后,

遍历该优先级队列中的节点, 依据反序搜索规则来计算出最短路径序列化图. 这 3 个算法的主要区别在于遍历优先级队列中节点的顺序有所不同.

4.1 LtS-NETDeg 算法

算法 2 描述了判断节点 u 和节点 v 之间是否存在可达路径的方法.

算法 2. $is\_existingpath(u,v,G)$.

```
1.  flag ← true
2.  EnQueue(Q,u)
3.  While flag do
4.    tempnode ← ExtractMin(Q)
5.    Foreach t ∈ nei(tempnode) do
6.      If t.sequence < tempnode.sequence or isBidirectional(t,tempnode) then
7.        If t == v then
8.          flag ← false
9.          break
10.       EndIf
11.     EnQueue(Q,t)
12.     EndIf
13.   EndFor
14.   If isEmpty(Q) then
15.     flag ← false
16.     break
17.   EndIf
18. EndWhile
19. If t == v then
20.   return true
21. else
22.   return false
23. EndIf
```

LtS-NETDeg (算法 3) 的输入为网络拓扑结构 G, 输出为节点的备份下一跳集合.在介绍算法 LtS-NETDeg 之前, 先介绍 2 个函数和 1 个算法. 函数 *permutation*(V) 是给图 G 中每个节点编号的函数, 它通过节点选择规则寻找满足要求的节点, 将它们按顺序加入到最后的输出序列中, 按顺序依次给输出序列中的节点编号, 编号就是节点的序号 (*sequence*), 使节点之间构成节点序列关系. 函数 *ExtractMin*(Q) 出队列的规则为, 选取度数的倒数最小的节点出队列, 如果有多个节点对应的度数的倒数相同, 则路由器 ID 最小的节点出队列. 算法 *is_existingpath*(u, v, G) 表示在图 G 中利用先进先出队列来实现寻找节点 u 到节点 v 之间的可达路径, 如果二者之间存在可达路径, 则返回 true, 否则返回 false. 算法 *is_existingpath*(u, v, G)(算法 2) 描述了计算节点 u 和节点 v 之间是否存在可达路径的过程. 将标志位 *flag* 初始化为 true, 创建一个先进先出队列 Q, 然后利用函数 *EnQueue*(Q, u) 将起始节点 u 加入队列 (算法 2 第 1, 2 行). 当 *flag* 为真时, 执行下面的循环操作. 循环开始时首先取出队列中的第 1 个元素, 并赋值给 *tempnode* (算法 2 第 4 行). 遍历 *tempnode* 的邻居节点 t, 如果节点 t 的序号小于 *tempnode* 节点的序号或者它们之间的边是双向边, 则将节点 t 加入队列 Q 中, 但如果遍历到的节点 t 就是节点 v, 则将 *flag* 置为 false, 并跳出循环 (算法 2 第 5–13 行). 如果队列 Q 为空, 则说明节点 u 和节点 v 之间没有可达路径, 将 *flag* 置为 false, 直接跳出循

环 (算法 2 第 14–17 行). 当循环结束以后, 如果节点 t 就是节点 v, 则返回 true, 否则返回 false (算法 2 第 19–22 行).

算法 3. LtS-NETDeg.

输入: $G=(V,E)$, 目的地址 d;
输出: $backn(u,d), u \in V\backslash\{d\}$.

```
1. permutation(V)
2. Foreach u ∈ V\{d} do
3.    Foreach v ∈ nei(u) do
4.       If v.sequence < u.sequence then
5.          backn(u,d) ← backn(u,d) ∪ {v}
6.       EndIf
7.    EndFor
8. EndFor
9. single_jump ← ∅
10. Foreach v ∈ V\{d} do
11.    If ||backn(v,d)|| == 1 then
12.       single_jump ← single_jump ∪ {v}
13.    EndIf
14. EndFor
15. Foreach v ∈ single_jump do
16.    EnQueue(Q,(v,1/deg(v),routerID(v)))
17. EndFor
18. G1 ← G
19. While isnotEmpty(Q) do
20.    u ← ExtractMin(Q)
21.    Foreach v ∈ nei(u) do
22.       If v ≠ bestn(u,d) then
23.          If not is_existingpath(v,v,G1) then
24.             If not is_existingpath(u,v,G1) then
25.                continue
26.             else
27.                G2 ← G1
28.                change_edge(u,v,G2)
29.                If is_existingpath(v,d,G2) then
30.                   backn(u,d) ← backn(u,d) ∪ {v}
31.                   G1 ← G2
32.                EndIf
33.             EndIf
34.          EndIf
35.       EndIf
36.    EndFor
```

37. **EndWhile**

38. **return** $backn(u,d), u \in V\backslash\{d\}$

下面将详细介绍 LtS-NETDeg 算法, 该算法是后面另外两个算法的基础. 对于网络中除去目的节点 d 以外的其他任意节点, 算法 3 第 1 行利用 *permutation*(V) 函数为节点编号. 如果该节点的邻居节点的序号小于该节点的序号, 则该邻居节点可以作为该节点到目的节点的备份下一跳 (算法 3 第 2–8 行). 初始化变量 *single_jump*, 将只有最优下一跳的节点存储在该变量中 (算法 3 第 9–14 行). 将 *single_jump* 中的节点加入到优先级队列 Q 中, 队列中的元素包括节点、节点度数的倒数和节点的路由器 ID (算法 3 第 16 行). 为了不影响原始网络拓扑结构, 将图 G 保存到图 $G1$ 中. 下面是一个循环过程, 循环结束的条件是队列为空. 当队列不为空时, 利用函数 *ExtractMin*(Q) 取出队列中的第 1 个节点 u. 遍历 u 的邻居节点 v, 如果该邻居节点不是节点 u 到节点 d 的最优下一跳, 则继续下面的计算. 如果邻居节点 u 没有自己到自己的可达路径, 但如果节点 v 有到达邻居节点 u 的路径, 则跳过该次循环继续遍历下个邻居节点 (算法 3 第 23–25 行), 否则, 继续下面的计算. 将图 $G1$ 保存到图 $G2$ 中, 将节点 u 和节点 v 之间的边修改为双向边, 如果在图 $G2$ 中节点 v 到达目的节点 d 有可达路径, 则节点 v 可以作为节点 u 到达目的节点 d 的备份下一跳, 将 $G2$ 赋值给 $G1$ (算法 3 第 27–31 行). 最后返回网络拓扑 $G1$ (算法 3 第 38 行).

4.2 StL-NETDeg 算法

StL-NETDeg 算法与 LtS-NETDeg 算法仅仅第 11 行不同, 优先级队列存储的是节点的度, 而不是节点度的倒数, 即按照节点度从小到大依次出队列.

4.3 Reduce-NodPri 算法

算法 4 描述了 Reduce-NodPri 算法的核心部分. 算法 Reduce-NodPri 与算法 LtS-NETDeg 的大部分执行过程是相同的, 下面仅介绍二者之间的区别. 初始化 *pre* 为目的节点 d, *pre* 记录当前出队列的节点 (算法 4 第 19 行). 在执行循环的过程中, 当某个节点出队列以后, 如果该节点等于 *pre* 节点, 则降低该节点的优先级, 将其优先级的数值加 1 (因为队列中优先级数值越低优先级越高, 所以数值加 1 说明节点的优先级降低), 这样可以防止出现相同节点连续出队列的情况 (算法 4 第 22, 23 行). 如果某个节点被遍历的次数超过 5 次, 则跳出循环 (算法 4 第 24–26 行). 每次循环过程中将 *pre* 更新为当前出队列节点 u (算法 4 第 44 行).

算法 4. Reduce-NodPri.

输入: $G = (V,E)$, 目的地址 d;

输出: $backn(u,d), u \in V\backslash\{d\}$.

1. $permutation(V)$
2. **Foreach** $u \in V\backslash\{d\}$ **do**
3. **Foreach** $v \in nei(u)$ **do**
4. **If** $v.sequence < u.sequence$ **then**
5. $backn(u,d) \leftarrow backn(u,d) \cup \{v\}$
6. **EndIf**
7. **EndFor**
8. $single\_jump \leftarrow \varnothing$
9. **EndFor**
10. **Foreach** $v \in V\backslash\{d\}$ **do**
11. **If** $\|backn(v,d)\| == 1$ **then**
12. $single\_jump \leftarrow single\_jump \cup \{v\}$
13. **EndIf**

14. **EndFor**
15. **Foreach** $v \in single\_jump$ **do**
16. $EnQueue(Q,(v,1/deg(v),routerID(v)))$
17. **EndFor**
18. $G1 \leftarrow G$
19. $pre \leftarrow d$
20. **While** $isnotEmpty(Q)$ **do**
21. $u \leftarrow ExtractMin(Q)$
22. **If** $u == pre$ **then**
23. $EnQueue(Q,(u,1/deg(v)+1,routerID(v)))$
24. **If** $visited(u) > 5$ **then**
25. **break**
26. **EndIf**
27. **EndIf**
28. **If** $v \neq bestn(u,d)$ **then**
29. **Foreach** $v \in nei(u)$ **do**
30. **If** $not\ is\_existingpath(v,v,G1)$ **then**
31. **If** $not\ is\_existingpath(u,v,G1)$ **then**
32. **continue**
33. **else**
34. $G2 \leftarrow G1$
35. $change\_edge(u,v,G2)$
36. **If** $is\_existingpath(v,d,G2)$ **then**
37. $backn(u,d) \leftarrow backn(u,d) \cup \{v\}$
38. $G1 \leftarrow G2$
39. **EndIf**
40. **EndIf**
41. **EndIf**
42. **EndFor**
43. **EndIf**
44. $pre \leftarrow u$
45. **EndWhile**
46. **return** $backn(u,d), u \in V \backslash \{d\}$

4.4 算法讨论

定理 3. 算法 LtS-NETDeg 和 StL-NETDeg 的时间复杂度为:

$$O((|V|-1)\lg|E|+|V|)+O(|V|+\lg|V|)+O(\min(|V|-1,\lg|V|)\times\lg|V|) \tag{13}$$

证明: 假设存放只有最优下一跳节点的优先级队列是 Q, 则算法 LtS-NETDeg 和 StL-NETDeg 的区别是优先级队列 Q 中存放节点优先级的规则不一样, 都是按照节点度作为优先级, 前者是按照节点度从大到小进行计算, 后者是按照节点度从小到大进行计算, 因此它们的时间复杂度一样. 公式 (13) 中第 1 项表示构造最短路径正序化图的时间复杂度, 其主要由两部分构成, 第 1 部分是从集合中选择最佳节点, 第 2 部分是给节点赋予序号; 第 2 项

表示获取拓扑中只有最优下一跳的节点的时间复杂度; 第 3 项中前半部分表示执行 $\min(|V|-1,\lg|V|)$ 次函数 $\arg\min_{hop_{[G]}^{\leqslant N(V)}(v)=1}(len(\mathrm{q}))$, 该函数使用深度优先搜索路径, 其时间复杂度为 $\lg|E|\mathrm{O}(|V|+|E|)$. 因此, 算法的时间复杂度为:

$$\mathrm{O}((|V|-1)\lg|E|+|V|)+\mathrm{O}(|V|+\lg|V|)+\mathrm{O}(\min(|V|-1,\lg|V|)\times\lg|V|).$$

定理 4. 算法 Reduce-NodPri 的时间复杂度为:

$$\mathrm{O}((|V|-1)\lg|E|+|V|)+\mathrm{O}(|V|+\lg|V|)+\mathrm{O}(\min(|V|-1,(\lg|V|+\max(hop_{[G]}^{\leqslant N(V)}(v)=1,\cdot\times5)))\lg|V|) \tag{14}$$

证明: 算法 Reduce-NodPri 与之前两个算法的区别在于使用了计算失败的节点优先级随遍历递减的规则, 公式 (14) 与公式 (13) 的区别在于第 3 项中前半部分遍历次数, $\min(|V|-1,(\lg|V|+\max(hop_{[G]}^{\leqslant N(V)}(v)=1,\cdot\times5)))$ 中 $(\lg|V|+\max(hop_{[G]}^{\leqslant N(V)}(v)=1,\cdot\times5))$ 表示遍历 $\lg|V|$ 的次数加上拓扑 G 中只有最优下一跳的节点被计算出来的最大次数 (不超过 5 次计算), $\cdot\times5$ 代表 $(hop_{[G]}^{\leqslant N(V)}(v)=1)\times5$, 即节点 v 被计算 5 次. 因此, 算法的时间复杂度为:

$$\mathrm{O}((|V|-1)\lg|E|+|V|)+\mathrm{O}(|V|+\lg|V|)+\mathrm{O}(\min(|V|-1,(\lg|V|+\max(hop_{[G]}^{\leqslant N(V)}(v)=1,\cdot\times5)))\lg|V|).$$

5 实验及结果分析

本节将对比算法 LtS-NETDeg、StL-NETDeg、Reduce-NodPri、DC、MARA-SPE 和 RPP-MTMAX 在平均备份下一跳数量、故障保护率和路径拉伸度这 3 个指标方面的性能.

5.1 网络拓扑

为了保证实验的准确性和可靠性, 本文采用 Rockerfuel 测量的真实拓扑[22]和 Brite[23]生成的模拟拓扑进行实验. 其中 Rockerfuel 测量的真实拓扑共有 17 个, 它们的节点数量的范围为 [11, 136], 链路数量的范围为 [21, 177], 具体参数如表 2 所示. Brite 软件的参数如表 3 所示.

表 2 Rockerfuel 测量的真实拓扑

拓扑结构	节点数量	链路数量
Agis	16	21
Ans	17	24
Arnes	31	43
Arpanet19719	18	22
Arpanet19723	24	27
Arpanet19728	29	32
Atmnet	21	22
AttMpls	25	56
Belnet2004	18	34
NJLATA	11	23
Renater2010	38	51
TataNld	136	177
TORONTO	25	55
USLD	28	45
VtlWavenet2008	88	92
Sunet	24	30
SwitchL3	30	51

表 3 Brite 拓扑生成器参数设置

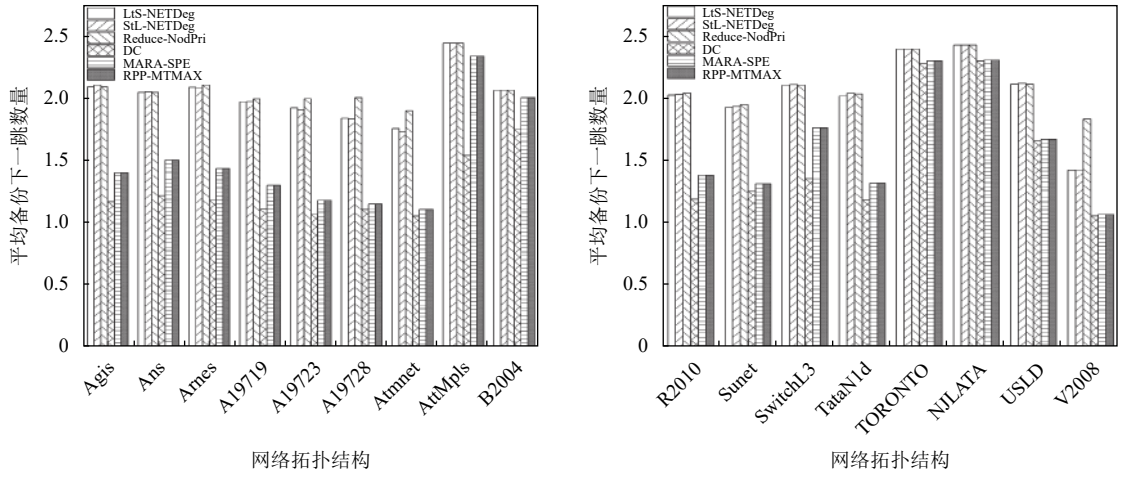
参数	设置
拓扑生成模型	Waxman
生成模式	路由器
链路代价	带宽的倒数
路由器数量	50–1 000
alpha	0.15
beta	0.2
节点平均度	2–12
链路带宽	10–1 024

5.2 评价指标

5.2.1 平均备份下一跳数量

图 5(a) 和图 5(b) 表示不同算法在真实拓扑中的实验结果. 可以看出, 在所有真实拓扑中, LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 的性能均优于 DC、MARA-SPE 和 RPP-MTMAX 的性能. 具体来说, 在 Agis、Ans、SwitchL3、TataNld 和 USLD 拓扑中, StL-NETDeg 的性能略优于 Reduce-NodPri; 而在其他网络拓扑中, Reduce-

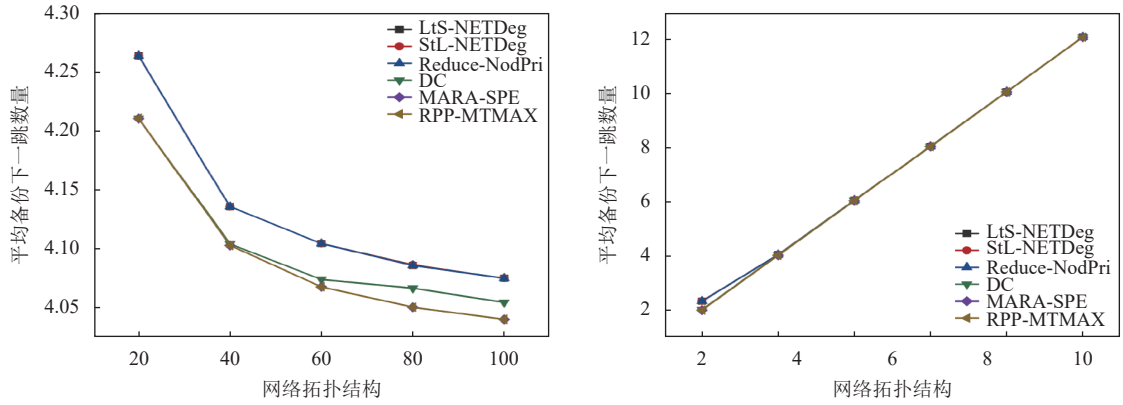
NodPri 的性能则更优. 在 AttMpls 中, LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 的性能表现最佳, 平均备份下一跳数量为 2.438 33; MARA-SPE 和 RPP-MTMAX 次之, 平均备份下一跳数量为 2.333 33; 而 DC 的平均备份下一跳数量仅为 1.538 33. 在 VtlWavenet2008 拓扑中, Reduce-NodPri 的性能明显优于其他 5 个算法, 平均备份下一跳数量为 1.827 46.



(a) 算法在 Agis 等 9 个网络拓扑的平均备份下一跳数量实验结果 (b) 算法在 R2010 等 8 个网络拓扑的平均备份下一跳数量实验结果

图 5 不同算法在真实拓扑上的平均备份下一跳数量实验结果

图 6(a) 和图 6(b) 显示了不同算法在模拟拓扑中的实验结果. 根据图 6(a), 随着网络拓扑规模的增大, 所有算法的平均备份下一跳数量都逐渐降低. 当网络拓扑规模为 20 时, LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 的性能表现最佳, 它们的平均备份下一跳数量都为 4.263 16. 根据图 6(b), 随着网络节点平均度数的增加, 所有算法的性能都有所提升, 且 6 种算法的性能差异逐渐减小. 例如, 当网络节点平均度数为 12 时, LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 算法的性能最佳, 平均备份下一跳数量为 12.065 75; 其次是 DC 算法, 平均备份下一跳数量为 12.065 33; 最后是 MARA-SPE 和 RPP-MTAMX, 平均备份下一跳数量为 12.060 3.



(a) 算法对应的平均备份下一跳数量和网络拓扑大小之间的关系 (b) 算法对应的平均备份下一跳数量和网络节点平均度之间的关系

图 6 不同算法在模拟拓扑上的平均备份下一跳数量实验结果

5.2.2 故障保护率

定义 6. 被保护节点. 对于网络中的任意节点 $u \in V$, 如果该节点的下一跳数量大于等于 2, 我们称该节点为被保护节点. 公式表示为:

$$hop_{[G]}(u) \geqslant 2 \tag{15}$$

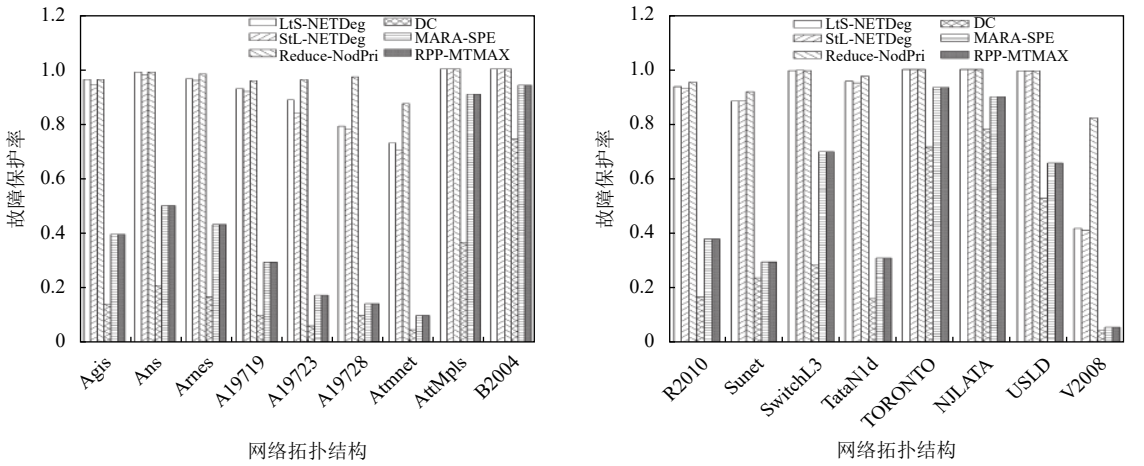
故障保护率 (failure protection ratio, Fr) 可以定义为:

$$Fr = \frac{\sum_{d \in V} N_d}{N(V)(N(V)-1)} \tag{16}$$

$$N_d = \sum_{u \in V} p(u) = \begin{cases} 1, & hop_{[G]}(u) \geqslant 2 \\ 0, & hop_{[G]}(u) < 2 \end{cases} \tag{17}$$

其中, N_d 是以节点 d 为目的节点时被保护节点的数量. $p(u)$ 用来判断节点 u 是否被保护.

图 7(a) 和图 7(b) 展示了不同算法在真实拓扑中的实验结果. 根据图 7(a) 和图 7(b), 在 Agis 拓扑中, Reduce-NodPri 和 LtS-NETDeg 的性能表现最佳, 故障保护率高达 0.962 5; 其次是 StL-NETDeg, 故障保护率为 0.941 67; 而 MARA-SPE 和 RPP-MTMAX 的故障保护率仅为 0.395 83; DC 的故障保护率最低, 仅为 0.141 67. 在 AttMpls、Belnet2004、TORONTO 和 NJLATA 中, Reduce-NodPri、LtS-NETDeg 和 StL-NETDeg 的故障保护率均达到了 1. 在所有真实拓扑中, DC 算法的性能表现最差, 而 Reduce-NodPri 的故障保护率基本都在 0.95 以上; 在 VtlWavenet-2008 拓扑中, 虽然 Reduce-NodPri 的故障保护率为 0.822 88, 但仍然明显优于其他 5 个算法.

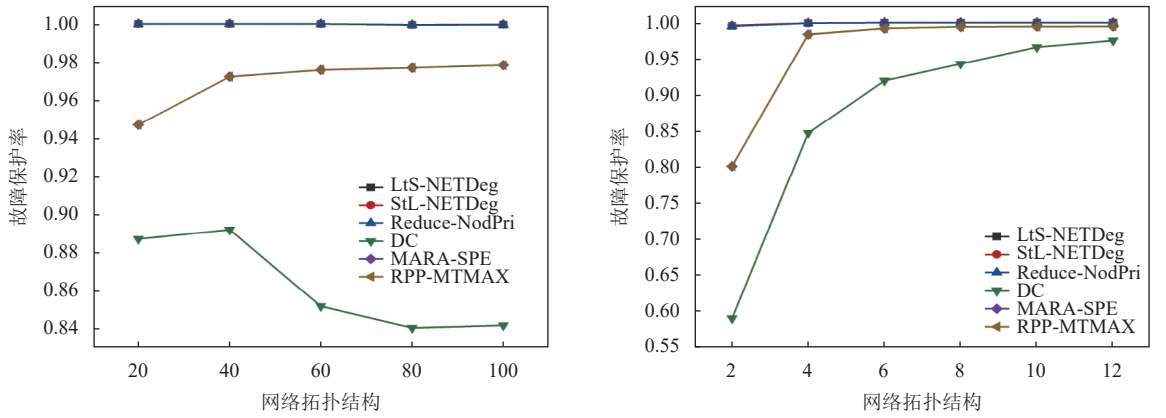


(a) 算法在 Agis 等 9 个网络拓扑的故障保护率实验结果　　(b) 算法在 R2010 等 8 个网络拓扑的故障保护率实验结果

图 7　不同算法在真实拓扑上的故障保护率实验结果

图 8(a) 和图 8(b) 表示不同算法在模拟拓扑中的实验结果. 根据图 8(a) 和图 8(b), 在图 8(a) 的 20、40、60、80、200 和图 8(b) 的 10、12 这 6 个拓扑中, Reduce-NodPri、LtS-NETDeg 和 StL-NETDeg 的故障保护率均达到了 1; 在其他模拟拓扑中, LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 的故障保护率也都在 0.99 以上, 远高于其他 3 种算法. 从图 8(a) 可以看出, 在图 8(a) 的 60 拓扑中, DC 的故障保护率为 0.851 69, MARA-SPE 和 RPP-MTMAX 的故障保护率为 0.975 99, 而 LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 的故障保护率均为 1. 从图 8(b) 可以看出, 随着网络节点平均度数的增加, 所有算法的性能都在逐渐提升, 但 Reduce-NodPri、LtS-NETDeg 和 StL-NETDeg 仍然保持最优的性能表现.

这是因为 DC 算法具有严格的约束条件, 其性能与拓扑结构密切相关. 而 MARA-SPE 和 RPP-MTMAX 仅考虑了节点之间的正序关系, 忽略了节点之间的反序关系. 相比之下, Reduce-NodPri、LtS-NETDeg 和 StL-NETDeg 则同时考虑了节点之间的正序关系和反序关系. 由于 Reduce-NodPri 算法是对 LtS-NETDeg 算法的进一步优化, 采用了多次节点重计算路由策略, 因此在实验结果中表现为 Reduce-NodPri 算法的性能优于或等于 LtS-NETDeg 算法. 此外, 随着网络度数的增加, 网络拓扑变得更加稠密, 节点的邻居节点数量增加, 算法计算出备份下一跳的概率也随之增加, 因此算法的性能普遍得到了提升.



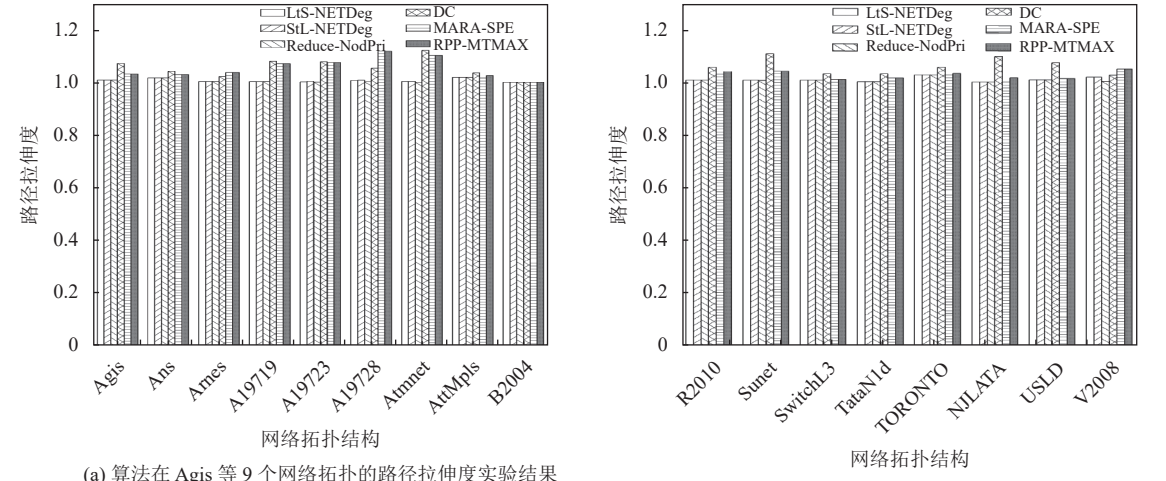
(a) 算法对应的故障保护率和网络拓扑大小之间的关系　(b) 算法对应的故障保护率和网络节点平均度之间的关系

图 8　不同算法在模拟拓扑上的故障保护率实验结果

5.2.3　路径拉伸度

路径拉伸度是衡量网络出现故障后, 备份路径代价与最短路径代价增加程度的重要指标. 具体来说, 路径拉伸度定义为受故障影响的源目的节点对的备份路径代价总和与这些节点对之间最短路径代价总和的比值. 它能够有效量化重路由路径与最优路由路径之间的差距, 反映网络路由成本的增加情况. 在网络故障情况下, 路由保护算法会生成新的重路由路径, 虽然这可以提高网络的稳定性, 但也可能增加数据包传输的成本, 导致网络资源的浪费. 因此, 在设计路由保护算法时, 应尽可能保证重路由路径的长度与最优路由路径的长度接近, 以减少额外的路由开销.

图 9(a) 和图 9(b) 显示了不同算法在真实拓扑中的实验结果. 根据图 9(a) 和图 9(b), 在大部分网络拓扑中, LtS-NETDeg、StL-NETDeg 和 Reduce-NodPri 的路径拉伸度均低于其他 3 个算法. 例如在图 9(a) 的 Atmnet 拓扑中, LtS-NETDeg、StL-NETDeg、Reduce-NodPri、MARA-SPE、RPP-MTMAX 和 DC 的路径拉伸度分别为 1.003 16、1.003 44、1.000 18、1.101 77、1.101 77 和 1.122 69. 在 Belnet2004 拓扑中, 6 个算法的路径拉伸度相同, 均为 1. 在图 9(b) 的 Sunet 拓扑中, DC 的路径拉伸度为 1.109 01, 而 Reduce-NodPri 的路径拉伸度仅为 1.005 88.

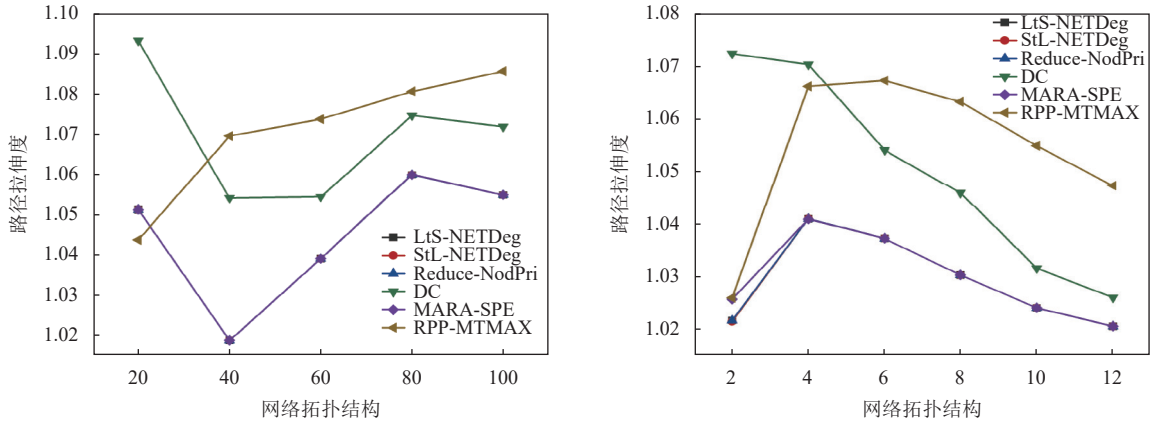


(a) 算法在 Agis 等 9 个网络拓扑的路径拉伸度实验结果　(b) 算法在 R2010 等 8 个网络拓扑的路径拉伸度实验结果

图 9　不同算法在真实拓扑上的路径拉伸度实验结果

图 10(a) 和图 10(b) 表示不同算法在模拟拓扑中的实验结果. 根据图 10(a) 和图 10(b), LtS-NETDeg、StL-NETDeg、Reduce-NodPri 的路径拉伸度明显低于其他 3 个算法, 这表明它们计算出的备份路由路径与最优路由路

径更为接近, 能够有效减少网络的额外开销. 例如在 200-2 拓扑中, MARA-SPE 的路径拉伸度为 1.072 26, 而 Reduce-NodPri 的路径拉伸度为 1.021 7. 在所有实验结果中, DC 和 RPP-MTMAX 的路径拉伸度均较高. 这是因为本文提出的算法可以计算出更多的备份下一跳数量, 当遇到故障时, 数据报转发过程中会优先选择开销最小的备份下一跳进行报文转发, 从而大大降低了额外开销.



(a) 算法对应的路径拉伸度和网络拓扑大小之间的关系　　(b) 算法对应的路径拉伸度和网络节点平均度之间的关系

图 10　不同算法在模拟拓扑上的路径拉伸度实验结果

6 总　结

为了提高网络中的路由可用性, 本文在反向最短路径树的基础上, 利用节点序列关系构建了最短路径正序化图, 并深入挖掘了反序搜索规则. 最终, 我们构造出了最短路径序列化图, 并基于此设计了域内路由保护算法. 实验结果表明, 该算法在平均备份下一跳数量、故障保护率和路径拉伸度等指标方面均表现出优越的性能. 本文提出的算法对路由可用性有明显的提升效果, 下一步的研究重点将是如何扩大其应用场景, 以进一步提高其应用价值和实用性.

References:

[1] Geng HJ, Shi XG, Wang ZL, Yin X, Hu ZG. Intra-domain routing protection scheme based on minimum intersection paths. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1536–1548 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5675.htm [doi: 10.13328/j.cnki.jos.005675]

[2] Chiesa M, Kamisinski A, Rak J, Retvari G, Schmid S. A survey of fast-recovery mechanisms in packet-switched networks. IEEE Communications Surveys & Tutorials, 2021, 23(2): 1253–1301. [doi: 10.1109/COMST.2021.3063980]

[3] Geng HJ, Shi XG, Wang ZL, Yin X. A hop-by-hop dynamic distributed multipath routing mechanism for link state network. Computer Communications, 2018, 116: 225–239. [doi: 10.1016/j.comcom.2017.12.008]

[4] Duijn IV, Jensen PG, Jensen JS, Krøgh TB, Madsen JS, Schmid S, Srba J, Thorgersen MT. Automata-theoretic approach to verification of MPLS networks under link failures. IEEE/ACM Trans. on Networking, 2022, 30(2): 766–781. [doi: 10.1109/TNET.2021.3126572]

[5] Akamai online retail performance report: Milliseconds are critical. 2017. https://www.akamai.com/newsroom/press-release/akamai-releases-spring-2017-state-of-online-retail-performance-report

[6] Qiu K, Zhao J, Wang X, Fu XM, Secci S. Efficient recovery path computation for fast reroute in large-scale software-defined networks. IEEE Journal on Selected Areas in Communications, 2019, 37(8): 1755–1768. [doi: 10.1109/JSAC.2019.2927098]

[7] Geng HJ, Zhang H, Shi XG, Wang ZL, Yin X. Efficient computation of loop-free alternates. Journal of Network and Computer Applications, 2020, 151: 102501. [doi: 10.1016/j.jnca.2019.102501]

[8] Foerster KT, Kamisinski A, Pignolet YA, Schmid S, Tredan G. Improved fast rerouting using postprocessing. IEEE Trans. on Dependable and Secure Computing, 2022, 19(1): 537–550. [doi: 10.1109/TDSC.2020.2998019]

[9] Jia XY, Li D, Zhu J, Jiang Y. Metro: An efficient traffic fast rerouting scheme with low overhead. IEEE/ACM Trans. on Networking,

2019, 27(5): 2015–2027. [doi: 10.1109/TNET.2019.2935382]

[10] Menth M, Hartmann M, Martin R, Čičić T, Kvalbein A. Loop-free alternates and not-via addresses: A proper combination for IP fast reroute? Computer Networks, 2010, 54(8): 1300–1315. [doi: 10.1016/j.comnet.2009.10.020]

[11] Anbiah A, Sivalingam KM. Efficient failure recovery techniques for segment-routed networks. Computer Communications, 2022, 182: 1–12. [doi: 10.1016/j.comcom.2021.10.033]

[12] Kvalbein A, Hansen AF, Čičić T, Gjessing S, Lysne O. Fast IP network recovery using multiple routing configurations. In: Proc. of the 25th IEEE Int'l Conf. on Computer Communications. Barcelona: IEEE, 2006. 1–11. [doi: 10.1109/INFOCOM.2006.227]

[13] Lakshminarayanan K, Caesar M, Rangan M, Anderson T, Shenker S, Stoica I. Achieving convergence-free routing using failure-carrying packets. ACM SIGCOMM Computer Communication Review, 2007, 37(4): 241–252. [doi: 10.1145/1282427.1282408]

[14] Yang Y, Xu MW, Li Q. Fast rerouting against multi-link failures without topology constraint. IEEE/ACM Trans. on Networking, 2018, 26(1): 384–397. [doi: 10.1109/TNET.2017.2780852]

[15] Zheng JQ, Xu H, Zhu XJ, Chen GH, Geng YH. We've got you covered: Failure recovery with backup tunnels in traffic engineering. In: Proc. of the 24th IEEE Int'l Conf. on Network Protocols. Singapore: IEEE, 2016. 1–10. [doi: 10.1109/ICNP.2016.7784449]

[16] Hu RQ, Geng HJ, Song YT. Study of routing availability framework based on node bias order relationship. Application Research of Computers, 2023, 40(4): 1160–1164, 1171 (in Chinese with English abstract). [doi: 10.19734/j.issn.1001-3695.2022.08.0431]

[17] Hartmann M, Hock D, Menth M. Routing optimization for IP networks with loop-free alternates. Computer Networks, 2016, 95: 35–50. [doi: 10.1016/j.comnet.2015.11.009]

[18] Csikor L, Tapolcai J, Rétvári G. Optimizing IGP link costs for improving IP-level resilience with loop-free alternates. Computer Communications, 2013, 36(6): 645–655. [doi: 10.1016/j.comcom.2012.09.004]

[19] Rétvári G, Tapolcai J, Enyedi G, Császár A. IP fast ReRoute: Loop free alternates revisited. In: Proc. of the 2011 IEEE INFOCOM. Shanghai: IEEE, 2011. 2948–2956. [doi: 10.1109/INFCOM.2011.5935135]

[20] Ohara Y, Imahori S, Van Meter R. MARA: Maximum alternative routing algorithm. In: Proc. of the 2009 IEEE INFOCOM. Rio de Janeiro: IEEE, 2009. 298–306. [doi: 10.1109/INFCOM.2009.5061933]

[21] Kwong KW, Gao LX, Guérin R, Zhang ZL. On the feasibility and efficacy of protection routing in IP networks. IEEE/ACM Trans. on Networking, 2011, 19(5): 1543–1556. [doi: 10.1109/TNET.2011.2123916]

[22] Spring N, Mahajan R, Wetherall D, Anderson T. Measuring ISP topologies with rocketfuel. IEEE/ACM Trans. on Networking, 2004, 12(1): 2–16. [doi: 10.1109/TNET.2003.822655]

[23] Heckmann O, Piringer M, Schmitt J, Steinmetz R. Generating realistic ISP-level network topologies. IEEE Communications Letters, 2003, 7(7): 335–336. [doi: 10.1109/LCOMM.2003.814708]

附中文参考文献:

[1] 耿海军, 施新刚, 王之梁, 尹霞, 胡治国. 基于最小路径交叉度的域内路由保护方案. 软件学报, 2020, 31(5): 1536–1548. http://www.jos.org.cn/1000-9825/5675.htm [doi: 10.13328/j.cnki.jos.005675]

[16] 胡睿乾, 耿海军, 宋艳涛. 基于节点偏序关系的路由可用性框架研究. 计算机应用研究, 2023, 40(4): 1160–1164, 1171. [doi: 10.19734/j.issn.1001-3695.2022.08.0431]

耿海军(1983－), 男, 博士, 副教授, 主要研究领域为软件定义网络, 路由算法, 网络体系结构.

胡治国(1977－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为网络测量, 路由算法.

胡睿乾(1999－), 男, 硕士生, 主要研究领域为路由算法, 网络体系结构.

尹霞(1972－), 女, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为下一代互联网, 协议测试.